Sistema de Archivos Simple

Trabajo Corto 4 – Principios de Sistemas Operativos

1st Pavel Zamora Araya
Escuela de Ingeniería en Computación
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
bzamora@estudiantec.cr

2nd Fabricio Solís Alpízar
Escuela de Ingeniería en Computación
Instituto Tecnológico de Costa Rica
Cartago, Costa Rica
dilsolis@estudiantec.cr

***Resumen*—This document presents the design, architecture, and implementation of a simple simulated filesystem developed in the C programming language as an Assignment of the Operating Systems Principles course. The project implements a 1 MB virtual storage medium, 512-byte block allocation, a root directory limited to 100 files, and core file operations including creation, reading, writing, deletion, and listing. The system adopts a modular architecture that separates storage handling, block management, directory structures, and file operations, all accessed through an interactive shell interface. Functional testing, error validation, stress testing, and fuzzing were carried out, demonstrating the robustness of the implementation and the educational value of understanding how a real file system manages data, metadata, and consistency under various workloads.**

I. Introducción

Los sistemas operativos incluyen subsistemas esenciales dedicados a diversas funcionalidades. Entre ellas, la gestión del almacenamiento, donde se encuentran mecanismos como la asignación de bloques, el control del espacio libre, el mantenimiento de directorios y la organización interna de los archivos. Comprender cómo funcionan estos componentes a bajo nivel es fundamental para el desarrollo de software de sistemas, administración de recursos y diseño de arquitecturas eficientes.

Este proyecto propone la creación de un sistema de archivos simulado que permita observar, analizar y comprender estos mecanismos desde cero. El simulador reproduce una estructura mínima similar a un sistema de archivos real, permitiendo ejecutar operaciones básicas que interactúan con un disco virtual estructurado en bloques.

II. Objetivos del Proyecto

Los objetivos principales del proyecto fueron:

- Implementar un sistema de archivos básico utilizando memoria virtual.
- Aplicar conceptos fundamentales de sistemas operativos como asignación de bloques, estructura de directorios y manejo de metadatos.
- Experimentar con validación, pruebas de errores y consistencia interna.
- Diseñar una arquitectura modular y mantenible en lenguaje C.

III. Descripción General

Esta sección presenta una visión general del funcionamiento del sistema implementado. El simulador utiliza un almacenamiento virtual de 1 MB dividido en bloques de 512 bytes. Esto genera un total de 2048 bloques disponibles para asignación. Se gestiona un directorio raíz con capacidad para almacenar la información de hasta 100 archivos simultáneamente.

Cada archivo contiene información sobre:

- su nombre,
- su tamaño lógico,
- la cantidad de bloques asignados,
- y la lista de bloques que ocupa en el disco simulado.

A continuación, se muestran las operaciones soportadas y una breve descripción de cada una:

- **CREATE**: crea un nuevo archivo y reserva los bloques necesarios.
- **WRITE**: escribe un texto en un offset específico dentro del archivo.
- **READ**: lee datos desde un offset determinado.
- **DELETE**: elimina un archivo, liberando sus bloques.
- **LIST**: muestra la información del sistema, incluyendo el espacio libre.

Estas operaciones representan el conjunto mínimo para la gestión de archivos dentro de un sistema operativo básico.

IV. Arquitectura Modular

Antes de describir los módulos, es importante resaltar el objetivo del diseño: crear una arquitectura de software clara, desacoplada y fácilmente extensible. La modularidad facilita la prueba individual de cada componente, reduce errores y permite agregar nuevas funcionalidades sin alterar la estructura existente.

El proyecto se estructura de la siguiente forma:

```
Filesystem-Simulator/
+-- storage.c        -> Disco virtual (1MB)
+-- block_manager.c  -> Bitmap de bloques
+-- directory.c      -> Directorio raíz
(100 archivos)
+-- file_operations.c-> CREATE/READ/WRITE/DELETE/
LIST
+-- filesystem.c     -> Orquestación del sistema
+-- main.c           -> Shell interactiva
```

Cada módulo se encarga exclusivamente de un área específica:

- **storage.c**: simula un disco de bytes contiguos.
- **block_manager.c**: administra los bloques libres y ocupados.
- **directory.c**: mantiene la información de los archivos.
- **file_operations.c**: implementa la lógica de las operaciones de archivo.
- **filesystem.c**: integra y coordina todos los módulos.
- **main.c**: provee un entorno interactivo tipo shell.

V. Estructuras de Datos

Esta sección explica las estructuras utilizadas para modelar el disco, el directorio y los archivos. La claridad en estas estructuras es esencial para un sistema de archivos confiable y eficiente.

V-A. Disco Virtual

El disco simulado se representa mediante un arreglo contiguo de bytes:

```
unsigned char storage[1048576];
```

V-B. Bitmap de Bloques

Para gestionar el espacio libre, cada bloque posee un indicador:

```
int block_used[2048];
```

V-C. Entrada del Directorio

Cada archivo en el directorio raíz se representa con la siguiente estructura:

```
typedef struct {
    char name[32];
    int size;
    int blocks[256];
    int block_count;
    int used;
} FileEntry;
```

V-D. Directorio Raíz

El directorio posee un máximo de 100 entradas:

```
FileEntry directory[100];
```

VI. Funcionamiento Interno

Esta sección detalla la lógica de cada operación ejecutada por el sistema, permitiendo observar cómo el simulador traduce comandos en acciones sobre el disco virtual.

VI-A. CREATE

La operación CREATE realiza:

1. Verificación de existencia del archivo.
2. Cálculo de los bloques requeridos según tamaño lógico.
3. Búsqueda de bloques libres (*first-fit*).
4. Actualización del directorio.

VI-B. WRITE

1. Valida existencia y límites.
2. Traduce offsets lógicos a ubicaciones físicas.
3. Escribe byte por byte en los bloques asignados.

VI-C. READ

1. Valida que el rango solicitado sea válido.
2. Traduce offsets.
3. Imprime los bytes solicitados.

VI-D. DELETE

1. Libera cada bloque del archivo.
2. Marca la entrada como no usada.

VI-E. LIST

1. Recorre el directorio.
2. Calcula el espacio libre revisando el bitmap.

VII. Decisiones de Diseño

Las principales decisiones de diseño llevadas a cabo incluyen:

- Asignación secuencial (*first-fit*) para minimizar la complejidad.
- Un único directorio raíz, evitando estructuras jerárquicas.
- Validación estricta de errores y parámetros.
- Separación clara entre almacenamiento, metadatos y operaciones.
- Estructuras de tamaño fijo para la estabilidad y eficiencia.

VIII. Pruebas y Validación

Esta sección detalla las distintas pruebas llevadas a cabo para asegurar el correcto funcionamiento del sistema.

VIII-A. Pruebas Funcionales

Incluyeron:

- Creación de archivos.
- Escritura en offsets variados.
- Lectura y verificación de datos.
- Eliminación completa y recuperación del espacio.

VIII-B. Pruebas de Errores

Se verificaron los siguientes escenarios:

- Offsets fuera de rango.
- Lectura de archivos inexistentes.
- Escrituras más allá del tamaño lógico declarado.
- Intentos repetidos de creación.

VIII-C. Fuzz Testing y Stress Testing

Se desarrolló un script que ejecutó más de 2000 operaciones aleatorias, validando que:

- No hubiera pérdidas de bloques.
- No quedaran archivos huérfanos.
- No ocurriera corrupción del directorio.
- El sistema siempre retornara a un estado consistente.

IX. Conclusiones

El sistema de archivos cumple completamente con los requisitos planteados. La arquitectura modular facilitó el diseño, la implementación y la depuración del sistema. Las pruebas exhaustivas demostraron la robustez de la solución y su consistencia ante diversas cargas de trabajo. Este proyecto permitió reforzar conceptos fundamentales de los sistemas operativos, como la asignación de bloques, la gestión del espacio libre y la consistencia de los metadatos, así como comprender mejor cómo estos sistemas llevan a cabo sus funciones y cómo se organizan internamente.

Agradecimientos

Los autores desean expresar su agradecimiento al Profesor Kenneth Obando Rodríguez por su guía, acompañamiento y explicaciones claras durante el desarrollo del proyecto. Su enfoque práctico permitió conectar los principios teóricos de los sistemas operativos con una implementación real en lenguaje C.

También, los autores reconocen el trabajo colaborativo del equipo, cuya comunicación efectiva y distribución equilibrada de tareas permitieron completar exitosamente el diseño, codificación, pruebas y documentación del sistema de archivos simulado.

Referencias

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, "Operating System Concepts," 10th Edition, John Wiley & Sons, Inc., 2018. [En línea]. Disponible: https://www.wiley.com/en-us/Operating+System+Concepts%2C+10th+Edition-p-9781119320913

[2] DevDocs, "C Language Reference," [En línea]. Disponible: https://devdocs.io/c/